# Efficient Optimization of Diameter and Average Shortest Path Length of a Graph using Path Count Index

Hiroshi Inoue

IBM Research – Tokyo

# My results in GraphGolf

## 2015

| | Rank | Author | Number of best solutions |
|---|---|---|---|
| 🏆 | 1 | Nobushimi & Ryo Ashida & Ryuhei Mori | 12 |
| | 2 | H. Inoue | 10 |
| | 3 | yawara & amami | 3 |

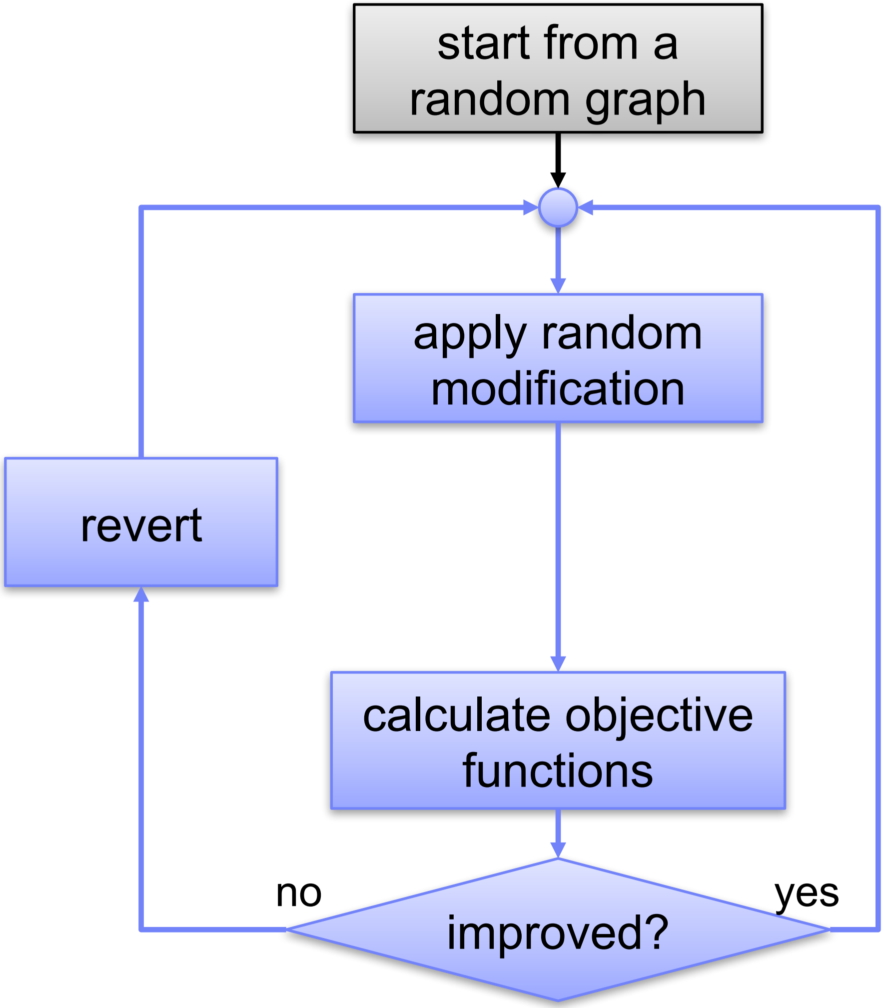
## 2016

| | Rank | Author | Number of best solutions |
|---|---|---|---|
| 🏆 | 1 | Takayuki Matsuzaki & Teruaki Kitasuka & Masahiro Iida | 10 |
| | 2 | H. Inoue | 6 |
| | 3 | Ryuhei Mori | 5 |

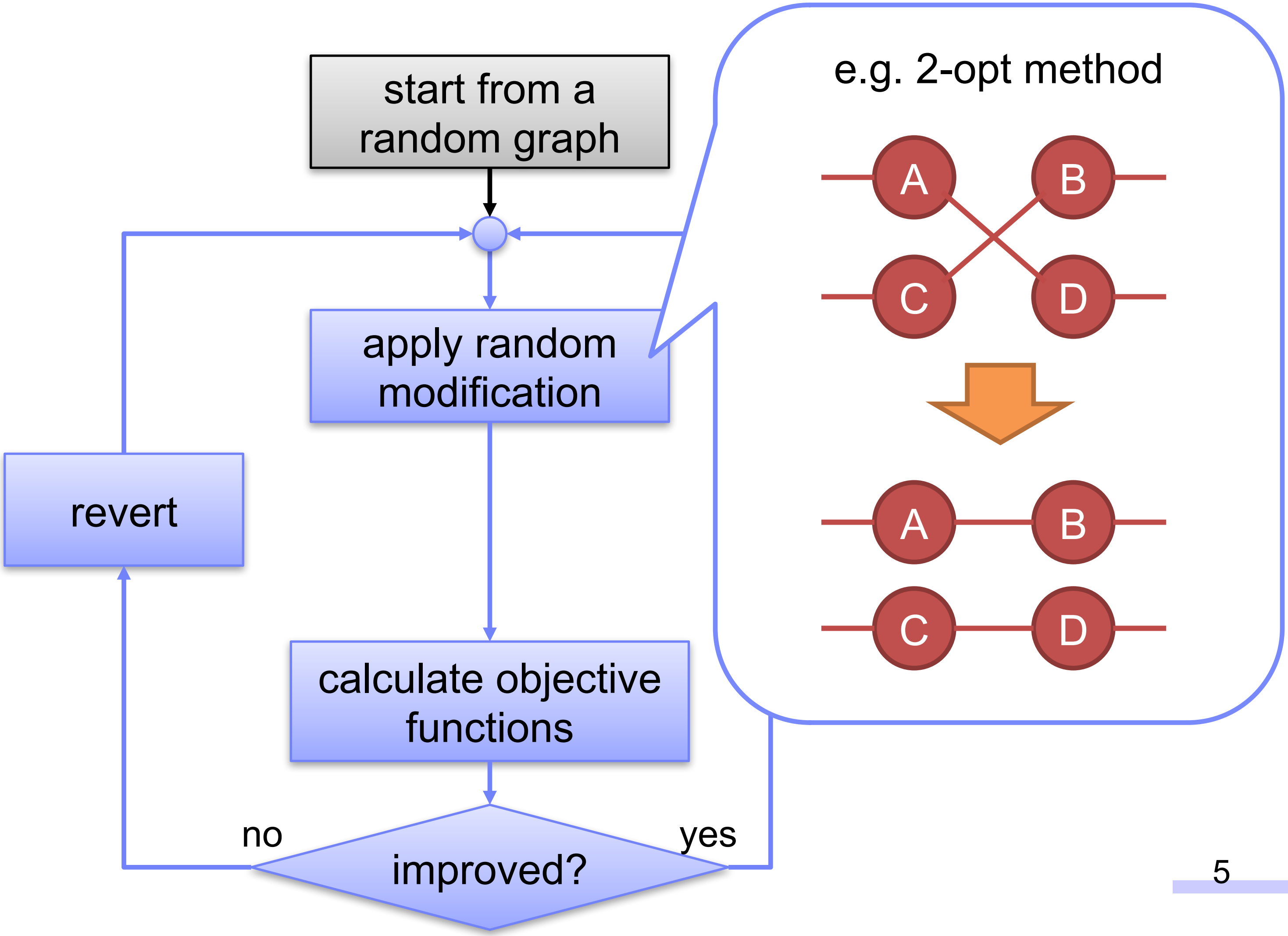
# Overview

- Problem to solve
  - Optimization problem of finding a graph with smaller diameter and ASPL (average shortest path length) for given order (number of nodes) ***n*** and degree ***d***
- My Approach
  - Local search with light-weight estimation of objective functions; naively calculation of objective functions is too costly for large graphs

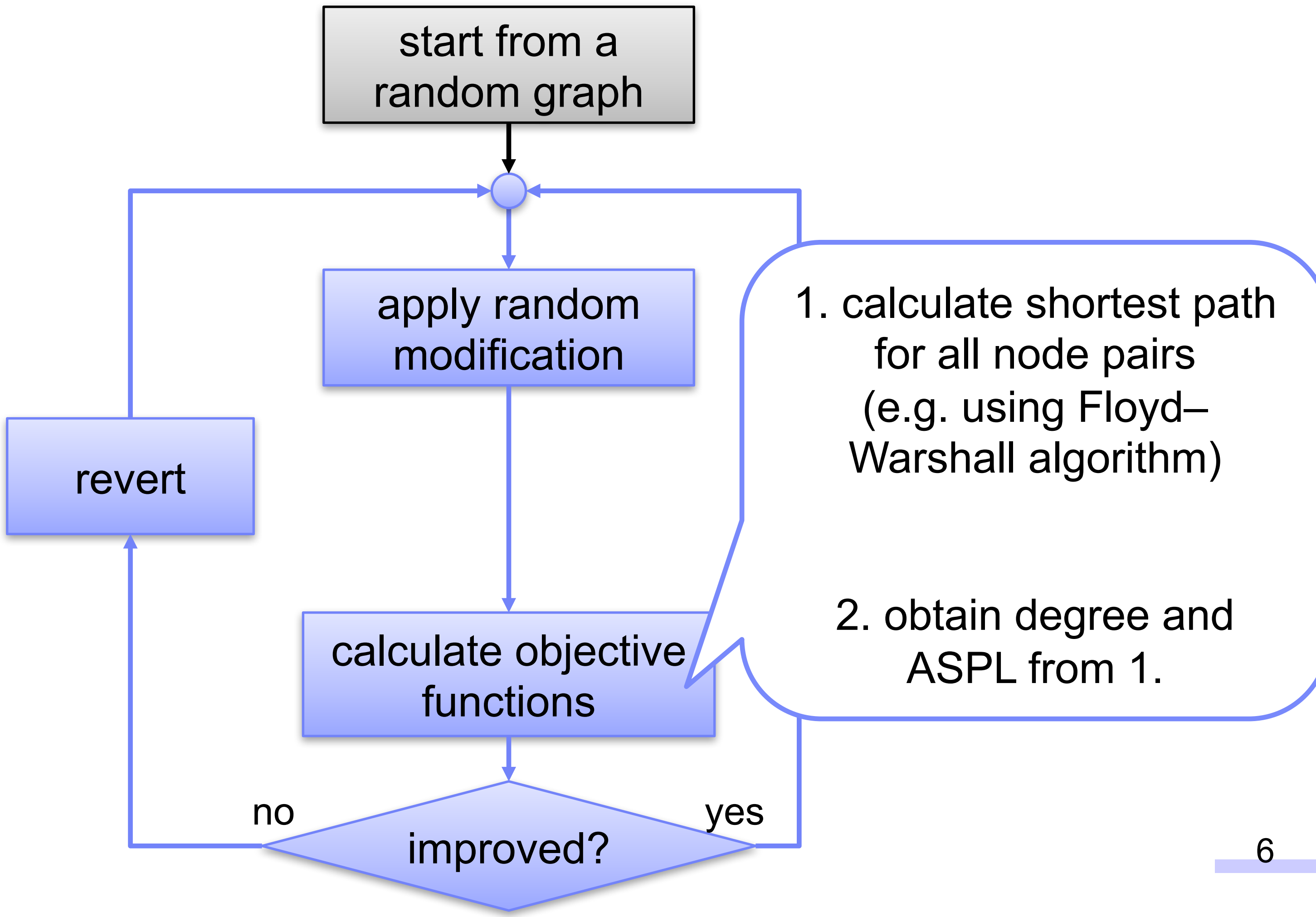
# Optimization process overview



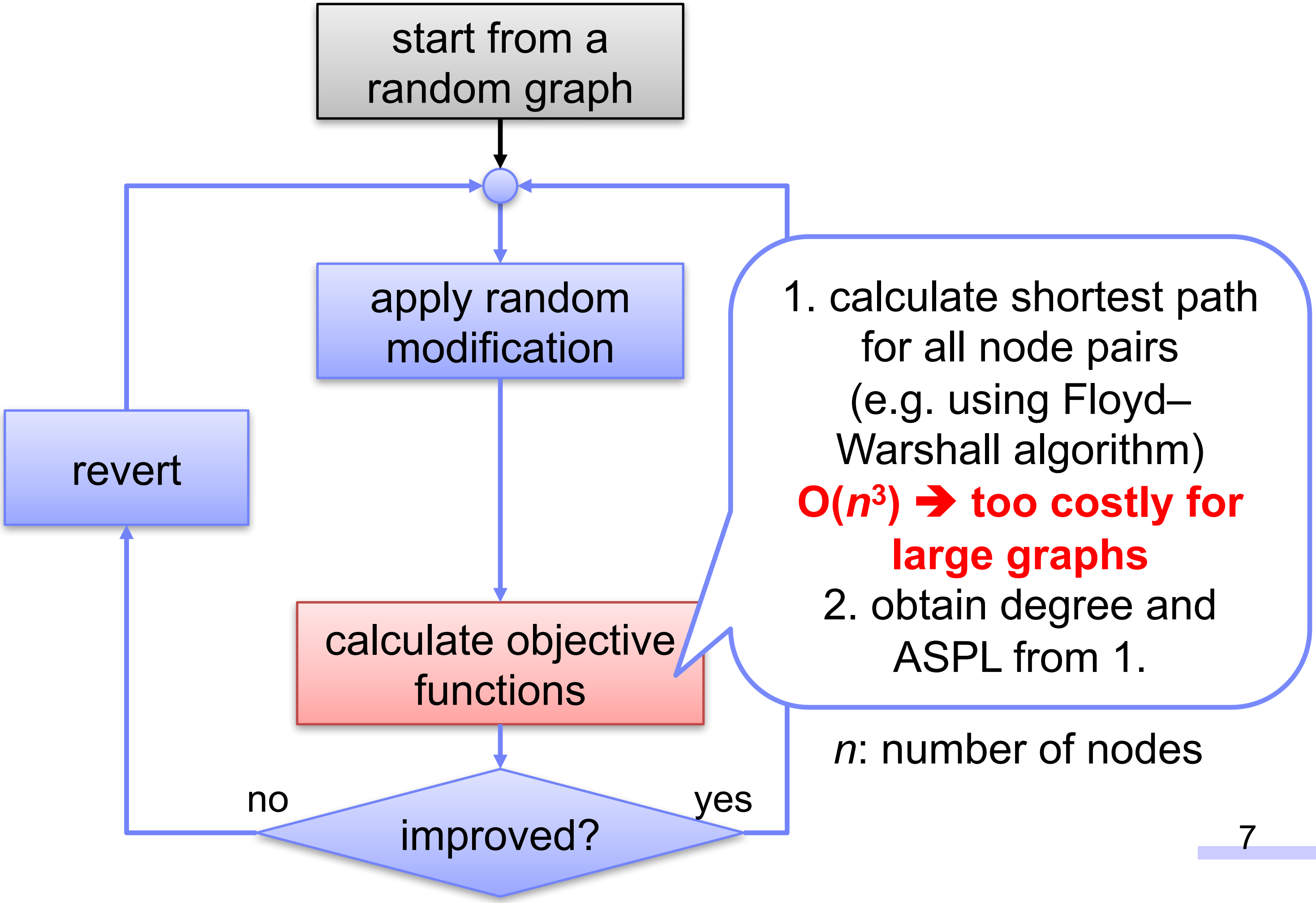
# Optimization process overview



# Optimization process overview

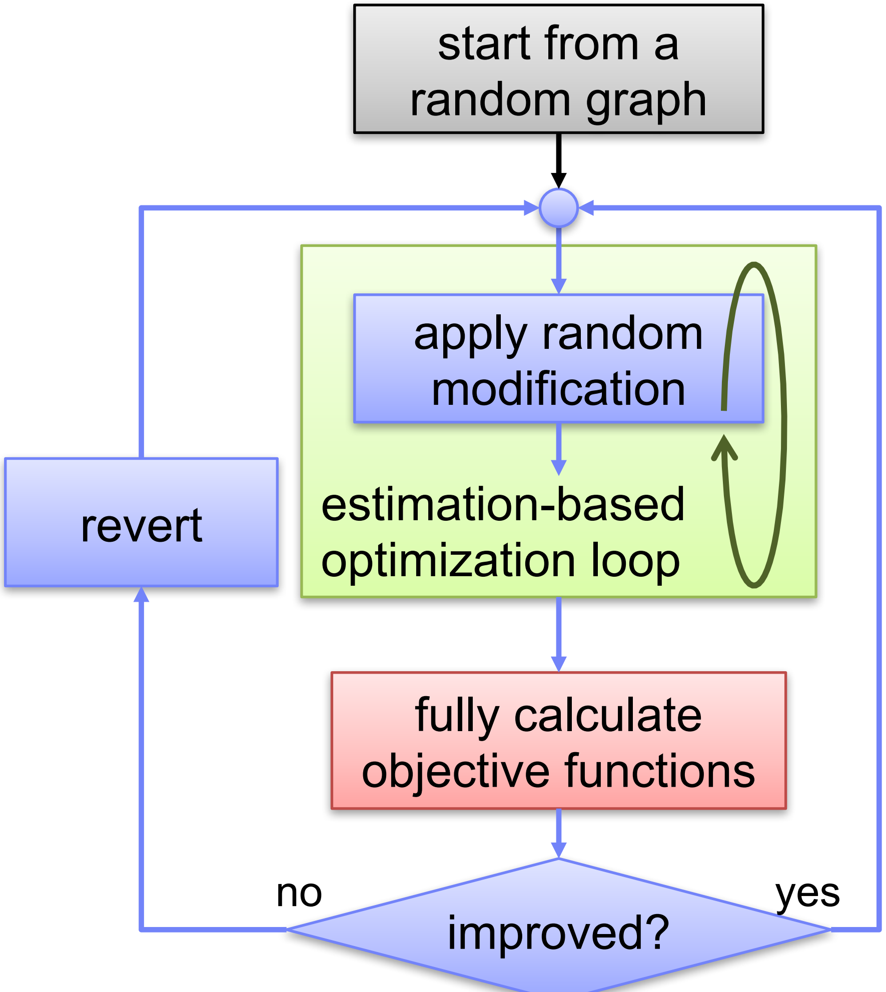


# Problem with large graphs

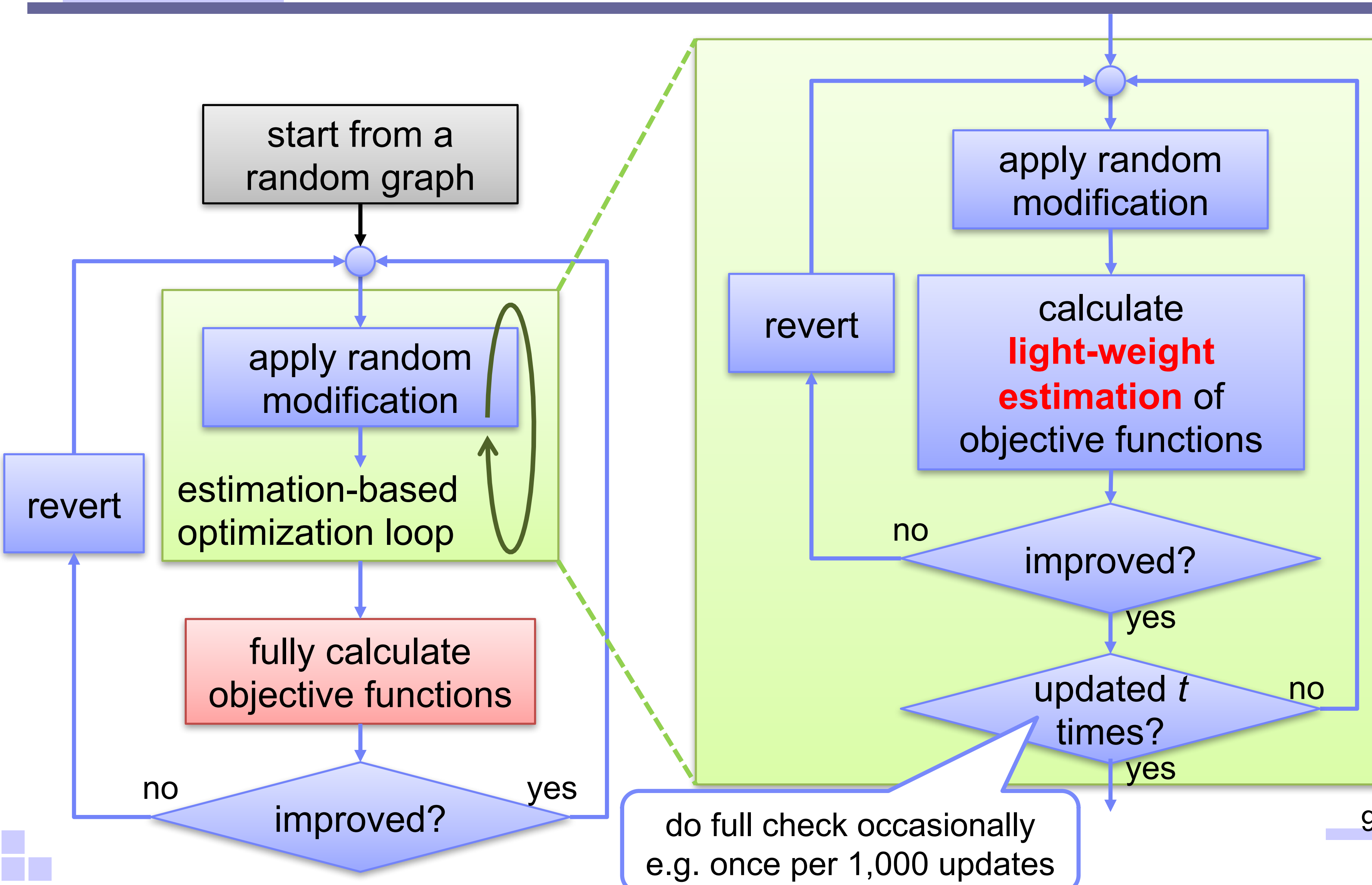



$n$: number of nodes

# Our approach



# Our approach



# How to estimate?

- what we actually need is:
  - not the current value of the objective functions (i.e. the diameter and ASPL)

    ☹ need to process the entire graph

    ➔ prohibitively costly for large graphs
  - but only the changes in the objective functions due to a small modification made in the graph

    ☺ can be calculated from the local information around the modified edges

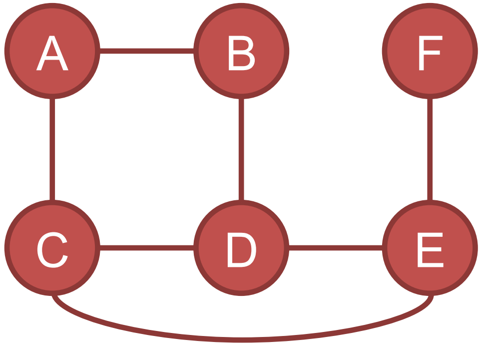
# Index to calculate changes

- To calculate the changes in all pairs shortest path when adding or removing an edge, we introduce a new index structure called *Path Count Index*

- Path Count Index is a lookup table that holds {*node1*, *node2*, *path length*} ➔ *number of paths*
  - $1 \leq path\ length \leq L_{max}$
  - if $L_{max} = 1$, Path Count Index is the adjacency matrix
  - excluding paths includes a cycle

# Example of path count index

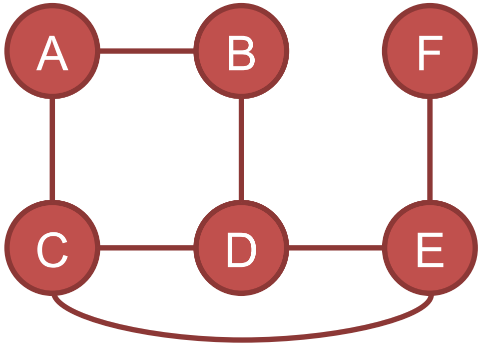
- { A, B, 1 } = 1 (A-B)
- { A, B, 2 } = 0
- { A, B, 3 } = 1 (A-C-D-B)
  - A-B-D-B and A-C-A-B are not counted




- { A, D, 1 } = 0
- { A, D, 2 } = 2 (A-B-D, A-C-D)
- { A, D, 3 } = 1 (A-C-E-D)

# Example of path count index

- { A, B, 1 } = 1 (A-B)
- { A, B, 2 } = 0
- { A, B, 3 } = 1 (A-C-D-B)
  - A-B-D-B and A-C-A-B are not counted

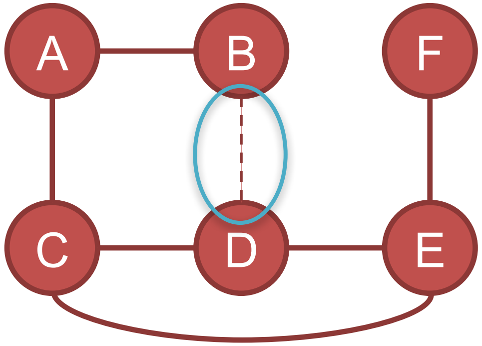



first non-zero entry = shortest path length

- { A, D, 1 } = 0
- { A, D, 2 } = 2 (A-B-D, A-C-D)
- { A, D, 3 } = 1 (A-C-E-D)

# Removing an edge (B→D)

- { B, D, 1 } = 1 → 0
- { A, D, 2 } = 2 → 1
- { B, C, 2 } = 2 → 1
- { B, E, 2 } = 1 → 0
- { A, C, 3 } = 1 → 0
- { B, E, 3 } = 2 → 1
- …

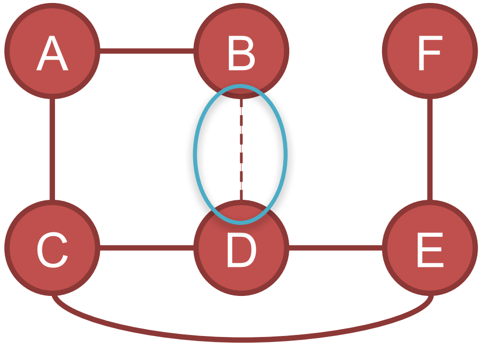



➔We calculate the changes in shortest path lengths while maintaining the path count index

# Removing an edge (B→D)

node B and E

- { B, E, 1 } = 0
- { B, E, 2 } = 1 → 0
- { B, E, 3 } = 2 → 1




shortest path length is increased by 1 (2→3)

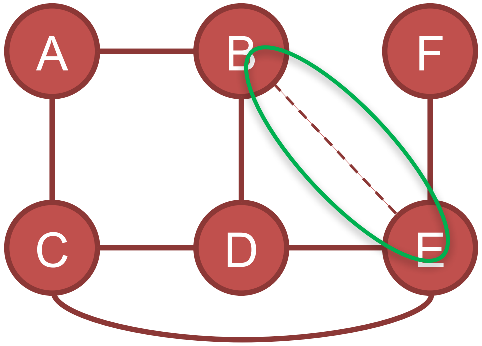
node A and D

- { A, D, 1 } = 0
- { A, D, 2 } = 2 → 1
- { A, D, 3 } = 1

shortest path length is not increased

# Adding an edge (B→E)

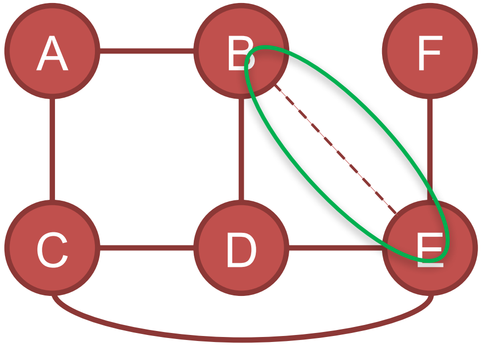
- { B, E, 1 } = 0 → 1
- { A, E, 2 } = 1 → 2
- { D, E, 2 } = 1 → 2
- { B, C, 2 } = 2 → 3
- { B, D, 2 } = 0 → 1
- { B, F, 2 } = 0 → 1
- { A, F, 3 } = 1 → 2
- …



# Adding an edge (B→E)

node B and F

- { B, F, 1 } = 0
- { B, F, 2 } = 0 → 1
- { B, F, 3 } = 1




shortest path length is decreased by 1 (3→2)

node A and F

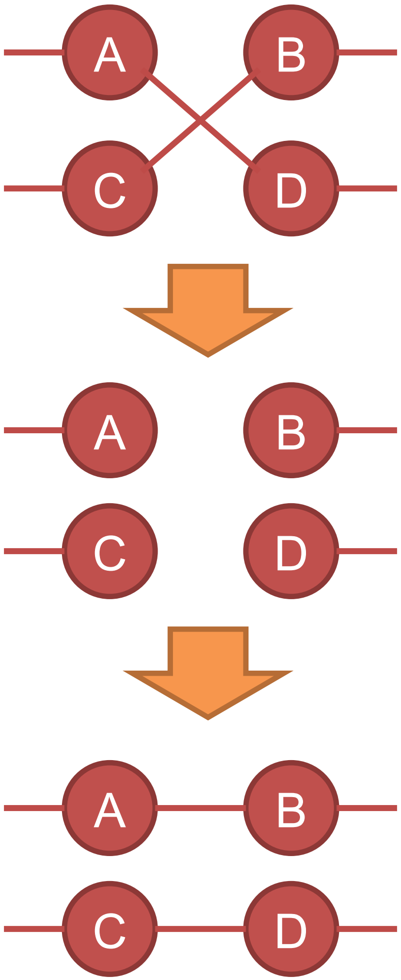
- { A, F, 1 } = 0
- { A, F, 2 } = 0
- { A, F, 3 } = 1 → 2

shortest path length is not decreased

# 2-opt and path count index

- One 2-opt step removes two edges and then adding two edges

➔ we can calculate changes in shortest path lengths (and hence ASPL and degree) as total of changes by four operations




- We can complete these operations only using local information without touching the entire graph

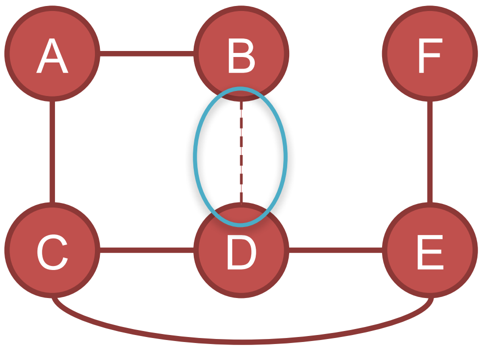
➔ efficient even for large graphs

# Why it gives only estimation?

- Because we have limitation in path length counted in path count index

node B and F (with $L_{max}$= 3)

- { B, F, 1 } = 0
- { B, F, 2 } = 0
- { B, F, 3 } = 1 → 0



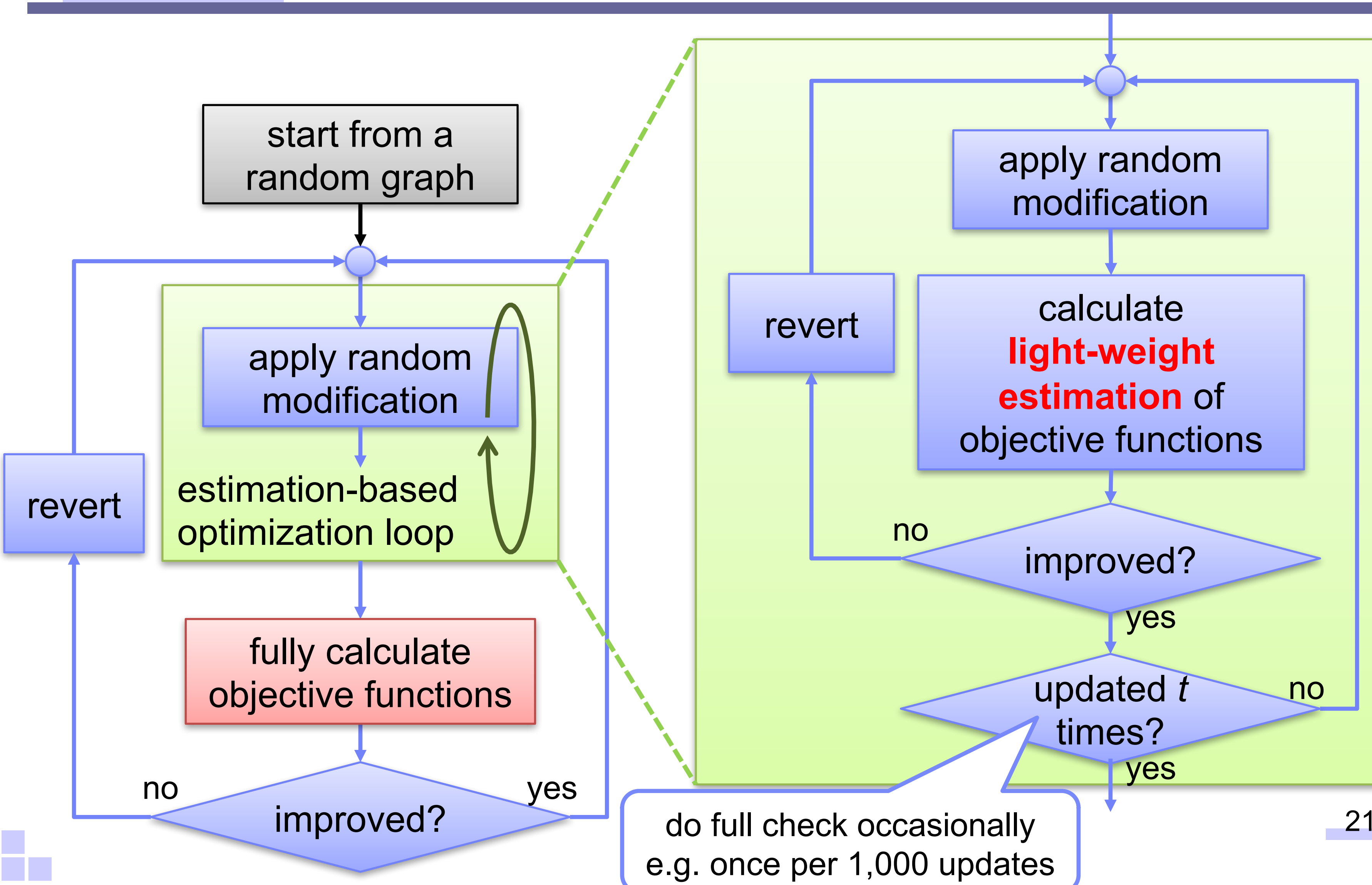

no non-zero entry for B-F → we assume $L_{max}$+1 is the shortest path length (this may incorrect!)

# How to decide $L_{max}$?

- $L_{max}$ is a parameter to control tradeoff between accuracy and performance
  - larger $L_{max}$ increases accuracy
  - smaller $L_{max}$ reduces overhead in memory size and computation cost
- $L_{max}$ equal to or slightly less than the diameter of the graph is a good choice for many cases

| (*n*, *d*) | current diameter | lower bound diameter | $L_{max}$ |
|---|---|---|---|
| 100k, 20 | 5 | 4 | 4 |
| 100k, 11 | 6 | 5 | 4 |
| 100k, 7 | 8 | 7 | 7 |
| 10k, 3 | 15 | 12 | 14 |

# Our approach



# metaheuristics

- based on (not-sophisticated) simulated annealing
  – if a better solution is not found after *T* trials (*T*: a pre-defined threshold), we accept a new solution that (slightly) worsen the objective functions

# Optimization in edge selection for 2-opt

- In graph, each edge has different relative importance on ASPL and degree (e.g. edge betweenness centrality)
- Randomly selecting two edges to remove in a 2-opt trial may remove an important edge

  → such 2-opt trial will not be successfully

# Sort-based edge selection for 2-opt

1. select $M$ edges randomly
2. for each edge, calculate the change (increase) in the objective function when removing the edge using path count index
3. sort $M$ edges by the changes
4. try 2-opt only for pairs selected from $m$ ($m < M$) edges having relatively low importance
   – e.g. $m = M / 4$ → the number of pairs becomes only 1/16 ($1/4^2$)

# Implementation

- Implemented in C++
- full computation of node-to-node distances employs pruned landmark labeling [1] library with minor modifications (https://github.com/iwiwi/pruned-landmark-labeling)
  - this library is not originally designed for all pairs distances computation
- (mostly) not parallelized!
  - only rarely-executed full computation of node-to-node distances is parallelized using OpenMP

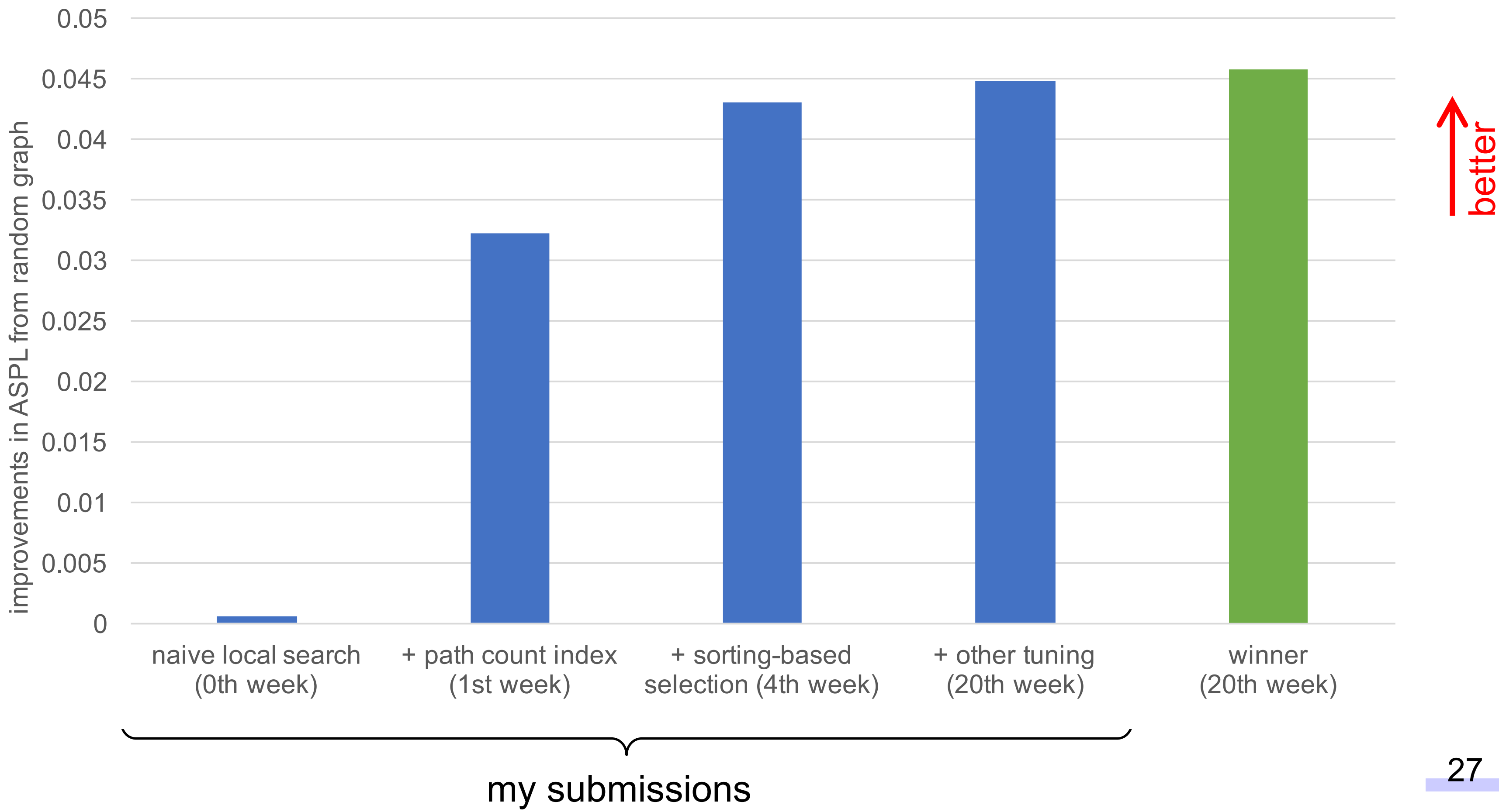
[1] Takuya Akiba et al., Fast exact shortest-path distance queries on large networks by pruned landmark labeling, SIGMOD '13

# Performance

- For large graphs (e.g. 100k-node configurations of GraphGolf 2016), the performance improvements are quite large since my algorithm reduces the computational complexity
  - naively computing all node-to-node distances of one graph ($n$=100k, $d$=20) takes more than 2 hours with 8 threads
  - one 2-opt trial with path count index takes about 0.1 sec (since we do not need to access the entire graph)
    ➔ speed up in order of $10^5$

# Performance

- From submission history of GraphGolf 2015 ($n$=10k, $d$=64)



# Memory consumption for path count index

- The number of paths of length *L* between two nodes is up to

$$\begin{cases} 1 & \text{for } L=1 \\ d \times (d-1)^{L-2} & \text{for } L>1 \end{cases}$$

- For example, the entry of path count index for one node pair with *d* = 3 and $L_{max}$ = 3 can fit in one byte
  - *L*=1 (up to 1 path): 1 bit
  - *L*=2 (up to 3 paths): 2 bits
  - *L*=3 (up to 6 paths): 3 bits

→ the total size of the path count index is

$$\frac{n \times n}{2} \times entry\_size$$

# Memory size optimization

- Problem:
  entry size may become too big for large $d$ and $L_{max}$
- e.g. for $d = 64$
  - $L$=1 (up to 1 path): 1 bit
  - $L$=2 (up to 64 paths): 6 bits
  - $L$=3 (up to 4032 paths): 12 bits
  - $L$=4 (up to 254k paths): 18 bits

# Memory size optimization

- Problem:
  entry size may become too big for large *d* and $L_{max}$
- e.g. for *d* = 64
  - *L*=1 (up to 1 path): 1 bit
  - *L*=2 (up to 64 paths): 6 bits → 4 bits
  - *L*=3 (up to 4032 paths): 12 bits → 4 bits
  - *L*=4 (up to 254k paths): 18 bits → 4 bits

we limit
up to 4 bits per *L*

➔ having the multiple paths of the same length is redundant and should not happen to achieve smaller ASPL

➔ in random graphs, large path counts in the path count index were rarely observed

# Total size of path count index for graph golf

| (*n*, *d*) | $L_{max}$ | entry size | total size | |
|---|---|---|---|---|
| 100k, 20 | 4 | 16 bit | 10 GB | |
| 100k, 11 | 4 | 16 bit | 10 GB | 2016 |
| 100k, 7 | 7 | 32 bit | 20 GB | |
| 10k, 64 | 2 | 8 bit | 50 MB | 2015 |
| 10k, 3 | 14 | 64 bit | 400 MB | |

- Note that, in the current implementation, we use 2x larger memory compared to the above size; we store the same data for (*i*, *j*) and (*j*, *i*) to avoid conditional branch overheads

# Reducing diameter

- In three categories, we won by a smaller diameter (not by a smaller ASPL)
  - $n$=1800/$d$=7 (2016), $n$=100k/$d$=11 (2016), $n$=4096/$d$=3 (2015)

1800 nodes, degree 7

| | Rank | Author | Diam. $k$ | ASPL $l$ | Diam. gap | ASPL gap | Improve | Info. | Date (UTC) | Week |
|---|---|---|---|---|---|---|---|---|---|---|
| ★ | 1 | H. Inoue | 5 | 4.07751 | 1 | 0.27596 | 0.02680 | | 2016-06-26 17:09:03 | 25 |
| | 2 | Nobushimi | 6 | 4.12909 | 2 | 0.32754 | 0.01323 | | 2016-06-24 04:32:02 | 25 |

➔ To reduce the diameter, we employ another objective function in the optimization

# Objective function for reducing diameter

- We focus on "number of node pairs whose distance is equal to the diameter" ($p$)
  - $p$ becomes 0 ➔ diameter is reduced by 1

- We can use $p$ as the objective function of the optimization instead of ASPL
  - base objective function: $100000k + l$
    ($k$: diameter, $l$ : ASPL)
  - objective function for diameter: $100000k +$ $\boldsymbol{p}$

- We can efficiently calculate $p$ from the path count index if $L_{max} \geq k$

# Entire optimization process

- Optimizing *p* typically worsen ASPL while optimizing ASPL (gradually) reduces *p*
- The entire optimization process using two objective functions are as follow:
  1. We start optimization for ASPL
  2. If *p* becomes relatively small (depends on graph sizem but typically less than 100~1,000), we manually switch the objective function for diameter based on *p*
  3. We go back to the normal objective function after getting a smaller diameter (i.e. *p* becomes 0)

# Diameter and ASPL

➔in some categories, I submitted two final solutions: one with best diameter and one with best ASPL

100000 nodes, degree 11

| | Rank | Author | Diam. $k$ | ASPL $l$ | Diam. gap | ASPL gap | Improve | Info. | Date (UTC) | Week |
|---|---|---|---|---|---|---|---|---|---|---|
| ★ | 1 | H. Inoue | 6 | 5.14685 | 1 | 0.28260 | 0.00392 | | 2016-06-26 17:11:18 | 25 |
| | 2 | H. Inoue | 7 | 5.14360 | 2 | 0.27934 | 0.00459 | | 2016-06-26 17:11:18 | 25 |

# Summary

- We introduced Path Count Index, which maintains the number of paths for all node pairs and path lengths
- We use Path Count Index for:
  - light-weight estimation of changes in shortest path length
  - efficient selection of target edges for 2-opt method
  - counting the number of node pairs having the distance equal to the diameter
- Since Path Count Index is a simple and flexible data structure, it will be potentially valuable for other operations with a dynamic graph

# Backup

# minor optimizations

- Try both of two ways of adding new edges in one 2-opt step
  - we need to pay the cost of edge removal only once for two trials